The Ford–Fulkerson max flow algorithm and the scaling method are still less than completely satisfactory, since the complexity depends on the capacities. It would be nice to have an algorithm whose asymptotic worst-case complexity is a small polynomial in m and n alone.

The following algorithm produces a max flow in time independent of the edge capacities. This algorithm was discovered independently by Dinic [2] and Edmonds and Karp [1]. It uses the following heuristic to achieve an $O(m^2n)$ running time:

$$\boxed{\text{Always choose an augmenting path of minimum length.}}$$

Define the *level graph* L_G of G to be the directed breadth-first search graph of G with root s with sideways and back edges deleted. The *level* of a vertex u is the length of a shortest path from s to u in G.

Note that the level graph has no edges from level i to level j for $j \geq i+2$. This says that any shortest path from s to any other vertex is a path in the level graph. Any path with either a back or sideways edge of the breadth-first search graph would be strictly longer, since it must contain at least one edge per level anyway.

Lemma 1

(a) *Let p be an augmenting path of minimum length in G, let G' be the residual graph obtained by augmenting along p, and let q be an augmenting path of minimum length in G'. Then $|q| \geq |p|$. Thus the length of shortest augmenting paths cannot decrease by applying the above heuristic.*

(b) *We can augment along shortest paths of the same length at most $m = |E|$ times before the length of the shortest augmenting path must increase strictly.*

Proof. Choose any path p from s to t in the level graph and augment along p by the bottleneck capacity. After this augmentation, at least one edge of p will be saturated (the bottleneck edge) and will disappear in the residual graph, and at most $|p|$ new edges will appear in the residual graph. All these new edges are back edges and cannot contribute to a shortest path from s to t as long as t is still reachable from s in the level graph. We continue finding paths in the level graph and augmenting by them as long as t is reachable from s. This can occur at most m times, since each time an edge in the level graph disappears. When t is no longer reachable from s in the level graph, then any augmenting path must use a back or side edge, hence must be strictly longer. □

This gives rise to the following algorithm:

Find the level graph L_G. Repeatedly augment along paths in L_G, updating residual capacities and deleting edges with zero capacity until t is no longer reachable from s. Then calculate a new level graph from the residual graph at that point and repeat. Continue as long as t is reachable from s.

With each level graph calculation, the distance from s to t increases by at least one by Lemma 1(a), so there are at most n level graph calculations. For each level graph calculation, there are at most m augmentations by Lemma 1(b). Thus there are at most mn augmentations in all. Each augmentation requires time $O(m)$ by DFS or BFS, or $O(m^2n)$ in all. It takes time $O(m)$ to calculate the level graphs by BFS, or $O(mn)$ time in all. Therefore the running time of the entire algorithm is $O(m^2n)$.

References

[1] J. Edmonds and R. M. Karp. Theoretical improvements in algorithmic efficiency for network problems. *J. Assoc. Comput. Mach.*, 19:248–264, 1972.

[2] E. A. Dinic. Algorithm for solution of a problem of maximal flow in a network with power estimation. *Soviet Math. Doklady*, 11:1277–1280, 1970.